A quick intro to running LArSoft

Karl Warburton with help from Tingjun Yang

02/02/2016

Guide to this guide

- A lot of the information shown here is taken from;
  - The dunetpc cheat sheet, which is here.
  - The LArSoft guide, which is here.
  - The 35 ton getting start guide, which is here.
  - The LArSoft concepts webpage, which is here.
  - An art/LArSoft course in June ‘15, which is here.
- LArSoft relies on the art framework which was developed by the Fermilab scientific computing division for intensity frontier experiments.
  - A useful (though HUGE) handbook to help use art can be found here.

Some steps to running LArSoft

- Basic running is very similar to NoVA-ART as they both depend on art and use Fermilab resources.

- FHiCL (fcl) and how to use it
- Where to save files
- Running on the batch
  - Job submission - how to do it, and limits
  - Project python
- Reconstruction in LArSoft
- Analysis in LArSoft

FHiCL

- Fermilab Hierarchial Command Language, used by a lot of experiments based at Fermilab.
- Some key feature in FHiCL files;
  - Previously defined configurations for modules and services are included with #include statements.
  - The services you want to include are defined in the services block, services:{ }
  - The source you want to use is defined in the source block, source: { }
  - The output you want to create is defined in the output block, outputs: { }
  - The physics you want to do is defined in the physics block, physics: { }

Some basic FHiCL rules

- You have to define a process name for each ‘job.’ This can’t be repeated (can’t run reco on the same file twice) and can’t contain things like _
- Parameter set names can’t contain numbers, ., /, * but may contain _
- All strings must be encompassed by “dfsf”
- Vectors are defined as [a, b, c]
- You pick out configurations from the include files using @local::< config name >
- You can override configurations in the fcl file with commands like
  - physics.producers.pmtrackdc.HitModuleLabel: "trkshowersplitdc”
  - The last value in the fcl file is the one used. By extension command line option take precedence over ones in fcl files.

Command line options

- Lar –h has more options but the most important are outlined in the NOvA-ART wiki

Executable and command line options

Currently there is one executable to run in NOvASoft. The executable to run a typical reconstruction or analysis job is nova which is placed in the user's path by the setup script. To see what options are available do

```
$nova -h
```

The output is

```
nova <options> [config-file]:
-T [ --TFileName ] arg File name for TFileService.
-c [ --config ] arg Configuration file.
-e [ --estart ] arg Event # of first event to process.
-h [ --help ] produce help message
-n [ --nevts ] arg Number of events to process.
--nskip arg Number of events to skip.
-o [ --output ] arg Event output stream file.
-s [ --source ] arg Source data file (multiple OK).
-S [ --source-list ] arg file containing a list of source files to read, one
per line.
--trace Activate tracing.
--notrace Deactivate tracing.
--memcheck Activate monitoring of memory use.
--nomemcheck Deactivate monitoring of memory use.
```

- Jobs are ran exactly the same as NOvA-ART, but using lar instead of nova.
  - lar -c prodsingle_dune35t.fcl

Example fcl file – prodsingle_dune35t

```
#include "services_dune.fcl"
#include "singles_dune.fcl"
#include "largeantmodules_dune.fcl"
#include "detsimmodules_dune.fcl"

process_name: SinglesGen

services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "single35t_hist.root" }
  TimeTracker:       {}
  RandomNumberGenerator: {} #ART native random number generator
  user:         @local::dune35t_simulation_services
}
#services.user.ExptGeoHelperInterface: @local::dune_geometry_helper
#services.user.Geometry.GDML: "dune35t4apa_v3.gdml"
#services.user.Geometry.ROOT: "dune35t4apa_v3.gdml"
#services.user.Geometry.SortingParameters.DetectorVersion: "dune35t4apa_v3"



#Start each new event with an empty event.
source:
{
  module_type: EmptyEvent
  timestampPlugin: { plugin_type: "GeneratedEventTimestamp" }
  maxEvents:   1           # Number of events to create
  firstRun:    1           # Run number to use for this file
  firstEvent:  1           # number of first event in the file
}

# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dune35t_singlep
   largeant:  @local::dune35t_largeant
   daq:       @local::dune35t_simwire
   rns:       { module_type: "RandomNumberSaver" }
   simcounter: @local::dune35t_simcounter
 }

 #define the producer and filter modules for this path, order matters,
 #filters reject all following items.  see lines starting physics.producers below
```

Example fcl file – prodsingle_dune35t

```
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ generator, largeant, daq, rns, simcounter ]
# simulate: [ generator, largeant, daq, rns]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}

#block to define where the output goes.  if you defined a filter in the physics
#block and put it in the trigger_paths then you need to put a SelectEvents: {SelectEvents: [XXX]}
#entry in the output stream you want those to go to, where XXX is the label of the filter module(s)
outputs:
{
 out1:
 {
   module_type: RootOutput
   fileName:    "single35t_gen.root" #default file name, can override from command line with -o or --output
 }
}
```

standard_reco_dune35t.fcl

```
#include "services_dune.fcl"
#include "caldata_dune.fcl"
#include "hitfindermodules_dune.fcl"
#include "cluster_dune.fcl"
#include "trackfindermodules_dune.fcl"
#include "pandoramodules_dune.fcl"
#include "calorimetry_dune35t.fcl"
#include "mctrutht0matching.fcl"
#include "t0reco.fcl"
#include "opticaldetectormodules_dune.fcl"
#include "photoncountert0matching.fcl"
#include "trackshowerhits.fcl"
#include "showerfindermodules_dune.fcl"
#include "emshower3d.fcl"

process_name: Reco

services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "reco_hist.root" }
  TimeTracker:       {}
  SimpleMemoryCheck:     { ignoreTotal: 1 } # default is one
  RandomNumberGenerator: {} #ART native random number generator
  message:      @local::dune_message_services_prod_debug
  FileCatalogMetadata:  @local::art_file_catalog_mc
  @table::dune35t_services
}
#source is now a root file
source:
{
  module_type: RootInput
  maxEvents:  10        # Number of events to create
}

# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
# random number saver
  rns:                { module_type: RandomNumberSaver }
# convert raw::RawDigit to recob::wire
  caldata:           @local::dune35t_calwire
# cheater reconstruction
```

standard_reco_dune35t.fcl

```
 mergeemshower3ddc:     @local::dune35t_mergeemshower3d
 blurredcluster:        @local::dune35t_blurredcluster
 emshower:              @local::dune35t_emshower
 emshower3d:            @local::dune35t_emshower3d
 mergeemshower3d:       @local::dune35t_mergeemshower3d
}

#define the producer and filter modules for this path, order matters,
#filters reject all following items.  see lines starting physics.producers below
reco: [ rns,
        #optical hit reco, flash, counter, TPC wire signals
        ophit, opflash, t0counter, caldata,
        #cheater reco
        hitcheat, clustercheat, trackcheat,
        #hit reco with cheated disambiguation
        gaushit, dcheat, fasthit,
        #cluster reco with cheated disambiguation
        dbclusterdc, lineclusterdc,
        #3D reco with cheated disambiguation
        costrkdc, mctrutht0dc, photont0costrkdc, calodc,
        #pandora with cheated disambiguation
        pandoradc, particlestitcherdc, mctrutht0pandoradc, photont0pandoradc, pandoracalodc,
        #pmatrack with cheated disambiguation
        pmtrackdc, mctrutht0pmtrackdc, photont0pmtrackdc, pmtrackcalodc,
        pmtrackpfpdc, mctrutht0pmtrackpfpdc, photont0pmtrackpfpdc, pmtrackpfpcalodc,
        #shower reconstruction
        blurredclusterdc, emshowerdc,
        #real disambiguation
        hit35t,
        #cluster reco
        dbcluster, linecluster,
        #3D reco
        costrk,  mctrutht0, photont0costrk, calo,
        #pandora
        pandora, particlestitcher, mctrutht0pandora, photont0pandora, pandoracalo,
        #pmatrack
        pmtrack, mctrutht0pmtrack, photont0pmtrack, pmtrackcalo,
        pmtrackpfp, mctrutht0pmtrackpfp, photont0pmtrackpfp, pmtrackpfpcalo,
        #shower reconstruction
        blurredcluster, emshower
]

#define the output stream, there could be more than one if using filters
stream1:  [ out1 ]

#trigger_paths is a keyword and contains the paths that modify the art::event,
#ie filters and producers
trigger_paths: [reco]
```

standard_reco_dune35t.fcl

```
#physics.producers.dcheatcc.ChanHitLabel:               "gaushit"
#physics.producers.fuzzydc.HitsModuleLabel:             "dcheat"
physics.producers.dbclusterdc.HitsModuleLabel:          "dcheat"
physics.producers.lineclusterdc.HitFinderModuleLabel:   "dcheat"
physics.producers.costrkdc.ClusterModuleLabel:          "lineclusterdc"
#physics.producers.stitchdc.TrackModuleLabel:           "costrkdc"
#physics.producers.stitchdc.SpptModuleLabel:            "costrkdc"
physics.producers.mctrutht0dc.TrackModuleLabel:         "costrkdc"
physics.producers.photont0costrkdc.TrackModuleLabel:    "costrkdc"
physics.producers.photont0costrkdc.HitsModuleLabel:     "lineclusterdc"
physics.producers.photont0costrkdc.ShowerModuleLabel:   ""
physics.producers.photont0costrkdc.TruthT0ModuleLabel: "mctrutht0dc"
physics.producers.calodc.TrackModuleLabel:              "costrkdc"
physics.producers.calodc.SpacePointModuleLabel:         "costrkdc"
physics.producers.calodc.T0ModuleLabel:                 "photont0costrkdc"
physics.producers.trkshowersplitdc.HitModuleLabel:       "lineclusterdc"
physics.producers.trkshowersplit.HitModuleLabel:        "linecluster"
physics.producers.pmtrack.HitModuleLabel:                "linecluster"
physics.producers.pmtrack.MakePFPs:                      true
physics.producers.pmtrackdc.HitModuleLabel:              "lineclusterdc"
physics.producers.pmtrackdc.ClusterModuleLabel:          "lineclusterdc"
physics.producers.pmtrackdc.MakePFPs:                    true
physics.producers.pmtrackcalodc.TrackModuleLabel:        "pmtrackdc"
physics.producers.pmtrackcalodc.SpacePointModuleLabel:   "pmtrackdc"
physics.producers.pmtrackcalodc.T0ModuleLabel:           "photont0pmtrackdc"
physics.producers.mctrutht0pmtrackdc.TrackModuleLabel:   "pmtrackdc"
physics.producers.photont0pmtrackdc.TrackModuleLabel:    "pmtrackdc"
physics.producers.photont0pmtrackdc.HitsModuleLabel:     "lineclusterdc"
physics.producers.photont0pmtrackdc.ShowerModuleLabel:   ""
physics.producers.photont0pmtrackdc.TruthT0ModuleLabel: "mctrutht0pmtrackdc"

physics.producers.pmtrackpfpdc.HitModuleLabel:              "lineclusterdc"
physics.producers.pmtrackpfpdc.ClusterModuleLabel:          "pandoradc"
physics.producers.pmtrackpfpdc.CluMatchingAlg:              3
physics.producers.pmtrackpfpdc.TrackingSkipPdg:             [0]
physics.producers.pmtrackpfpdc.RunVertexing:                true
physics.producers.pmtrackpfpdc.FlipToBeam:                  true
physics.producers.pmtrackpfpdc.MakePFPs:                    true

physics.producers.pmtrackpfpcalodc.TrackModuleLabel:        "pmtrackpfpdc"
physics.producers.pmtrackpfpcalodc.SpacePointModuleLabel:   "pmtrackpfpdc"
physics.producers.pmtrackpfpcalodc.T0ModuleLabel:           "photont0pmtrackpfpdc"
physics.producers.mctrutht0pmtrackpfpdc.TrackModuleLabel:   "pmtrackpfpdc"
physics.producers.photont0pmtrackpfpdc.TrackModuleLabel:    "pmtrackpfpdc"
physics.producers.photont0pmtrackpfpdc.HitsModuleLabel:     "lineclusterdc"
physics.producers.photont0pmtrackpfpdc.ShowerModuleLabel:   ""
physics.producers.photont0pmtrackpfpdc.TruthT0ModuleLabel: "mctrutht0pmtrackpfpdc"
```

Where to save things I

- nashome/p/user
  - Not sure about space, gets backed up, but not mounted on the gird machines.
  - I for one do nothing here…
- /dune/app area
  - Each user has 200 GB, but IS NOT designed to hold data.
  - Fill up semi-regularly if people save a lot of data there.
  - Is backed up ‘snapshot’ every day (I think).
- /dune/data/ and /dune/data2/
  - Each user has 200 GB on each, IS designed to hold data.
  - Is good for analysis files that you definitely want to keep hold off.
- Only the app areas are mounted on the grid.

Where to save things II

- /pnfs/dune/persistent/
  - Unlimited storage, basically forever but if it fills up, gets close, or you're using lots of space people will chase you.
  - Good for the output of batch jobs you are likely to want to keep.
- /pnfs/dune/scratch
  - Unlimited storage, but only lasts ~1 month.
  - Good for the output of batch jobs you are unlikely to want to keep forever.
- DO NOT MOVE BETWEEN PERSISTENT AND SCRATCH
  - It will keep the properties it had before, ie a scratch file moved to persistent will be deleted after a month. You need to copy things to prevent this.
- Neither are mounted on the grid, you must copy files you want to process to the worker node – scp.

Job submission

- The Fermilab grid uses the jobsub client, the basic commands you will want to use are outlined below.

Command	Description
jobsub_submit	submits jobs
jobsub_submit_dag	submits jobs with dependencies on the execution order
jobsub_q --user=trj	queries trj's jobs. Use instead of condor_q trj. Gives you job ID's
jobsub_rm --jobid=<jobid>	kills jobs
jobsub_fetchlog -J <jobid>	fetches log files from the fifebatch servers
jobsub_fetchlog --list-sandboxes	get a list of fetchable job id's (available in jobsub_client v1_0_3 or later)

- To run things on the grid as dune you must be on the DUNE VO. This should be done by default, but if not then either submit a service desk ticket, or ask Tom Junk.
- An example of how to submit two jobs is below.

```
source /grid/fermiapp/products/dune/setup_dune.sh
setup jobsub_client
jobsub_submit -N 2 -M --OS=SL6 --group=dune --resource-provides=usage_model=OPPORTUNISTIC file:///dune/app/users/trj/batchprobe
```

Job submission

- Using the –N XX option
  - Using the PROCESS variable can tell your shell script to do specific things depending on which jobId it has.
- Storing things whilst on the grid
  - Use the _CONDOR_SCRATCH_DIR
    - Make a directory _CONDOR_SCRATCH_DIR/work and ifdh cp files there. Then ifdh cp finished files once the job is completed.
- The default memory limit is 2000 MB, you can request more with the --memory=2048 option.
- You can request more time with –-expected_lifetime
- Jobs which exceed memory or lifetime will be held.

Example script to submit jobs

```
#!/bin/sh

umask 002

odir=${_CONDOR_SCRATCH_DIR}/work
mkdir -p $odir
export HOME=$odir
logdir=${_CONDOR_SCRATCH_DIR}/log
mkdir -p $logdir
export LOG=$logdir

echo $HOME
echo $LOG

FILENUMBER=$(($PROCESS+1))


source /dune/app/users/jti3/mytest/job/mysetup.sh


cd /dune/app/users/jti3/mytest/job


ifdh cp -D /dune/app/users/jti3/mytest/job/slicedfiles.txt  $odir/

sed -n "${FILENUMBER}{p;q;}" $odir/slicedfiles.txt > $odir/inputFile
read -r line < $odir/inputFile
echo $line

ifdh cp -D $line $odir/

ls $odir/lbne* > $odir/newfile
read -r line2 < $odir/newfile


lar -c RunSSPReco_filtered.fcl -s $line2 -T $odir/SSPOutput_job_${PROCESS}.root >& $logdir/SSPOutput_${PROCESS}.log

echo DONE RUNNING

ifdh cp -D $odir/SSPOutput_job_${PROCESS}.root /pnfs/lbne/persistent/users/jti3/ophits/output8/
ifdh cp -D $logdir/SSPOutput_${PROCESS}.log /pnfs/lbne/persistent/users/jti3/ophits/logs8/

echo DONE COPYING

exit
```

Project python

- A python script which takes away a lot of the hassle of submitting jobs – in a similar vain to submit_nova_art.py from what I can tell.
- There are two wiki's covering it.
  - One from larbatch which explains the structure really well.
  - One in dunetpc which is more of a functional description of how to use it.
- Either submit on the command line
  - project.py –h
  - projectgui.py –-xml < XML > &

Reconstruction in LArSoft

- There was a protoDUNE working meeting at the end of June ‘16. Lots of presentations about reconstruction.
  - Tingjun – Overview
  - Xin – Signal processing
  - Xin – Wire cell
  - Robert – Pattern recognition
  - Joris – Pandora
  - Dorota – PMATrack
- As it is the most general I am going to go through Tingjuns talk, but I encourage you to look at the others.

Analysis in dunetpc

- There is as yet no default analysis chain for dunetpc.
- There are some of mentions of trying to use something similar to HIGHLAND which T2K uses.
- Currently there is an analysis module which makes a flat ROOT TTree which aims to be the base of art independent analyses.
  - It is a very complicated module which aims to maximize memory usage so it can run of the cluster.
  - dunetpc/dune/AnaTree/
    - HowToUseAnalysisTree.txt – description
    - AnalysisTree_module.cc - module